Volume 4, Issue 2

Methods

Mouser's technology & solutions journal

Understanding A.I.

Creating Programs That Learn
p. 3

Open-source Movement Affects AI Apps
p. 33

AI's Evolution Demands Strong Ethics
p. 37

In this issue

3 **Creating Programs That Learn**
by Stephan Evanczuk

9 **Machine Learning Requires Multiple Steps**
by M. Tim Jones

15 **Agency, Autonomy, and Protection in AI**
by Sally Eaves

21 **Living with an Imperfectly Ethical AI**
by Michelle Nedashkovskaya

25 **Machine-Learning Software Simplifies Development**
by Stephan Evanczuk

33 **Open-source Movement Affects AI Apps**
by Jim Romeo

37 **AI's Evolution Demand Strong Ethics, Safety**
by Kyle Dent

Methods

Mouser's technology & solutions journal

Contributing Authors

Stephan Evanczuk
M. Tim Jones
Sally Eaves
Michelle **Nedashkovskaya**
Jim Romeo
Kyle Dent

Technical Contributors

Paul Golata
Joseph Downing
Christina Unarut

Design & Production

Robert Harper

Special Thanks

Kevin Hess
Sr. VP, Marketing

Russell Rasor
VP, Supplier Marketing

Jack Johnston, Director
Marketing Communication

Raymond Yin, Director
Technical Content

Creating Programs That Learn

By Stephan Evanczuk for Mouser Electronics

Artificial intelligence lies at the heart of dramatic advances in automotive, healthcare, industrial systems, and an expanding number of application areas. As interest continues to rise, the nature of AI has elicited some confusion and even fear about the growing role of AI in everyday life. The type of AI that enables an increasing number of smart products builds on straightforward but nontrivial engineering methods to deliver capabilities far removed from the civilization-ending AI of science fiction.

Definitions of AI range from its most advanced—and still conceptual—form, where machines are human-like in behavior, to a more familiar form where machines are trained to perform specific tasks. In its most advanced form, true artificial intelligences would operate without the explicit direction and control of humans to arrive independently at some conclusion or take some action just as a human might. At the more familiar engineering-oriented end of the AI spectrum, machine-learning (ML) methods typically provide the computational foundation for current AI applications. These methods generate responses to input data with impressive speed and accuracy without using code explicitly written to provide those responses. While software developers write code to process data in conventional systems, ML developers use data to teach ML algorithms such as artificial neural network models to generate desired responses to data.

How is a basic neural network model built?

Among the most familiar types of machine learning, neural network models pass data from their input layer through hidden layers to an output layer (**Figure 1**). As described, the hidden layers are trained to perform a series of transformations that extract the features needed to distinguish between different classes of input data. These transformations culminate in values loaded into the output layer, where each output unit provides a value representing the probability that the input data belongs in a particular class. With this approach, developers can classify data such as images or sensor measurements using an appropriate neural network architecture.

Neural network architectures take many forms, ranging from the simple type of feedforward neural network shown in **Figure 1** to deep neural networks (DNNs) built with several hidden layers and individual layers containing hundreds of thousands of neurons. Nevertheless, different architectures typically build on an artificial neuron unit with multiple inputs and a single output (**Figure 2**).

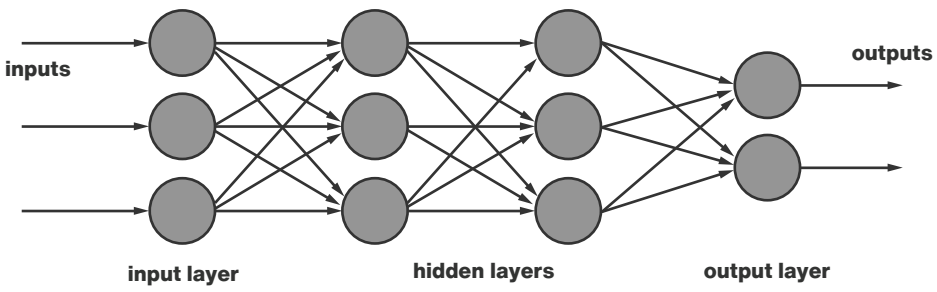

Figure 1: *Neural networks comprise layers of artificial neurons trained to distinguish between different input data classes. (Source: adapted from Wikipedia)*


AI

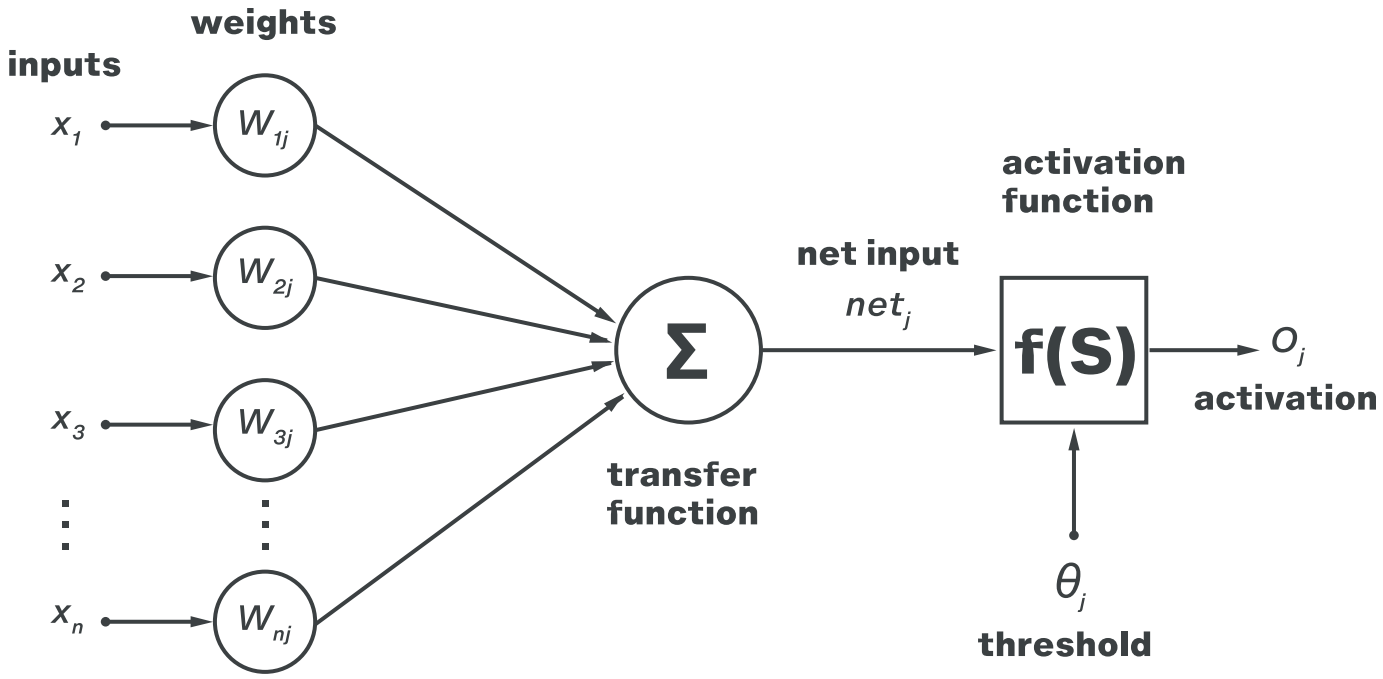

Figure 2: *An artificial neuron produces an output based on an activation function that operates on the sum of the neuron's weighted inputs. (Source: Wikipedia)*

In a feedforward neural network, a particular neuron n_{ij} in hidden layer j sums its i inputs, x_i, adjusted by an input-specific weight w_{ij}, and adds a layer-specific bias factor b_j (not shown in the figure) as follows:

$$S_j = \Sigma w_{ij} x_i + b_j$$

Finally, the summed value S_j is converted to a single value output by an activation function. Depending on requirements, these functions can take many forms, such as a simple step function, arc tangent, or non-linear mapping such as a rectified linear unit (ReLU), which outputs 0 for $S_j <= 0$ or S_j for $S_j > 0$.

Although they are all designed to extract the distinguishing features of data, different architectures might use significantly different transformations. For example, convolutional neural networks (CNNs) used in image-recognition applications use kernel convolutions. In this, functions, called kernels, perform convolutions on the input image to transform it into feature maps. Subsequent layers perform more convolutions or other functions, further extracting and transforming features until the CNN model generates a similar classification probability output as in simpler neural networks.

However, for developers, the underlying math for popular neural network architectures is largely transparent because of the availability of ML development tools (discussed elsewhere in this issue). Using those tools, developers can fairly easily implement a neural network model and begin training it using a set of data called the training set. This training data set includes a representative set of data observations and the correct classification for each observation—and represents one of the more challenging aspects of neural network model development.

How is a neural network model trained and deployed?

In the past, developers creating training sets had little option but to work through the many thousands of observations required in a typical set, manually labeling each observation with its correct name. For example, to create a training set for a road sign recognition application, they need to view images of road signs and label each image with the correct sign name. Public domain sets of prelabeled data let many machine-learning researchers avoid this task and focus on algorithm development. For production ML applications, however, the labeling task can present a significant challenge. Advanced ML developers often use pre-trained models in a process called transfer learning to help ease this problem.

Although emerging tools and services help facilitate data preparation, the training set characteristics nevertheless play a critical role in the effectiveness of the neural network model and overall application. The decisions made in choosing which observations to include and exclude have fundamental implications, including flexibility, specificity, and fairness that require careful consideration. As a result, the level of effort required to create an optimal training set can rival the effort required to implement the machine-learning program itself. After the training set is created and the neural network model is implemented, the model's training process iteratively runs the training data. At each iteration, the training process calculates a loss function that measures the difference between the desired result provided by the data labels and the calculated result generated by the model. Using a method called back propagation, that error information is used by the training process to adjust the weights and other model parameters for the next iteration. This process continues until the loss function falls within some threshold or fails to improve after some specified number of iterations.

When training completes, the model is converted to an inference model by performing several optimizations, including removing unneeded structures such as the back propagation mechanism, eliminating neurons that contribute little to the classification process, and even merging layers. Programs implement the inference model by loading a compact representation saved in various standard formats by ML tools and frameworks.

Neural networks not always the best solution

Although neural networks might be the more recognizable type of machine learning, they are by no means the only or even best choice for some applications. Neural networks fall into ML called supervised learning because they rely on labeled data sets to train the algorithm. In sensor-based applications such as the Internet of Things (IoT) or industrial systems, other supervised learning algorithms like support vector machines (SVMs) or decision trees provide an alternative that is simpler, more compact, and equally effective.

SVM methods classify data by finding where input data points lie within an n-dimensional space defined by the training data. Decision-tree methods use training data to construct a model that efficiently decomposes input data into a series of optimized decisions. As with the output layer of a neural network, the decision tree's final leaf nodes provide the probability that the data falls into a particular class. This approach is particularly efficient for classifying sensor data such as simultaneous accelerometer and gyroscope measurements to detect a complex movement. Support for decision trees is integrated into some inertial measurement units from STMicroelectronics.

These supervised learning methods are perhaps the most recognizable ML form. Still, other types of ML, including unsupervised learning, reinforcement learning, and many others, are already being applied to practical engineering problems. As the name suggests, unsupervised learning finds relationships within unlabeled data, using clustering techniques to identify similar characteristics. These techniques are particularly useful during training set development and feature engineering, where developers optimize the selection of data characteristics or features to be used for training and inference. Reinforcement learning finds applications in robotics systems programming, using utility measures to optimize training, not unlike loss functions in neural network training.

Conclusion

Machine-learning methods such as neural networks form the foundation of a growing array of smart products able to recognize and classify specific images or sensor measurements based on sophisticated mathematical concepts. For developers, implementing neural network models and other ML-based solutions in their applications follows a well-supported development process that is straightforward but by no means simple. M

Kria™ KV260 Vision AI Starter Kit

mouser.com/xilinx-kria-kv260-kit

XILINX®

Machine Learning Requires Multiple Steps

By M. Tim Jones for Mouser Electronics

Introduction

Deploying machine learning (ML) is a multi-step process. It involves selecting a model, training it for a specific task, validating it with test data, and then deploying and monitoring the model in production. Here, we'll discuss these steps and break them down to introduce you to ML.

ML refers to systems that, without explicit instruction, are capable of learning and improving. These systems learn from data to perform a particular task or function. In some cases, learning, or more specific training, occurs in a supervised manner where incorrect outputs result in adjusting the model to nudge it toward the correct output. In other cases, unsupervised learning occurs where the system organizes the data to reveal previously unknown patterns. Most ML models follow these two paradigms (supervised vs. unsupervised learning).

Let's now dig into what is meant by a model and then explore how data becomes the fuel for machine learning.

Machine-Learning Model

A model is an abstraction of a solution for machine learning. The model defines the architecture which, once trained, becomes an implementation. Therefore, we don't deploy models. We deploy implementations of models trained from data (more on this in the next section). So models plus data plus training equal instances of ML solutions (**Figure 1**).

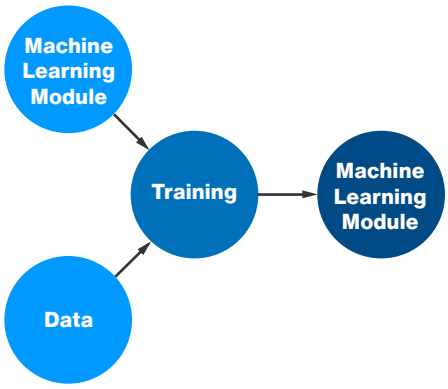

Figure 1: *From Machine Learning Model to Solution. (Source: Author)*

ML solutions represent a system. They accept inputs, perform the computation of different types within the network and then provide an output. The input and output represent numerical data which means that, in some cases, translation is required. For example, feeding text data into a deep-learning network requires encoding words into a numerical form that is commonly a high-dimensional vector given various words that could be used. Similarly, outputs might require translation from a numerical form back into a textual form.

ML models come in many types, including neural network models, Bayesian models, regression models, clustering models, and more. The model that you choose is based upon the problem at hand.

In the context of neural networks, models range from shallow multi-layer networks to deep neural networks that include many layers of specialized neurons (processing units). Deep neural networks also have a range of models available based upon your target application. For example:

- If your application is focused on identifying objects within images, then the Convolutional Neural Network (CNN) is an ideal model. CNNs have been applied to skin-cancer detection and outperform the average dermatologist.

ADuCM4050 Ultra Low Power Microcontroller

mouser.com/adi-aducm4050-ulp-mcu

ANALOG DEVICES

AHEAD OF WHAT'S POSSIBLE™